实验报告一：Multiboot启动

报告人：PB16071259 张润邦

实验内容：

本实验是Multiboot启动实验，旨在编写一个支持Multiboot启动协议的helloworld内核，并使用GRUB(+虚拟机)或直接使用qemu启动helloworld内核。

实验原理：

首先需了解Multiboot启动协议。操作系统启动一般都需要一个boot loader来加载，不同的操作系统可能需要不同的boot loader来启动。Multiboot启动协议则规定了boot loader与操作系统的接口规范，任何满足该协议的boot loader都可加载任一满足该协议的操作系统。满足Multiboot规范(i386)的boot loader会将让CPU进入保护模式，可访问所有的内存。

qemu是一个满足协议的可执行硬件虚拟化的开源托管虚拟机。本实验中使用qemu启动OS kernel，要求OS内核使用ELF格式，包含一个Multiboot header。header的内容最主要的是三个4字节数: magic (等于0x1BADB002)、flags (设为0)、checknum (用作校验)。

helloworld的输出结果分别通过VGA和串口输出。VGA接口是显卡上的一个接口，本实验VGA显存的起始地址为：0xB8000，每传输一个字符需要2个字节表示，一个确定字符本身，一个确定字符的显示属性。串口采用异步收发传输器，发送数据时，CPU将并行数据写入UART，UART按照一定的格式在一根电线上串行发出。本实验中串口起始地址为0x3F8。

源代码：

multibootHeader.S

```
.globl start        #声明入口符号

.section .multiboot_header  #自定义一个Header段
        .long 0x1BADB002              #magic
        .long 0x00                    #flag
        .long - (0x1BADB002 + 0x00)   #checknum
.code32

.data       #定义一个数据段，用来声明待输出的字符串
    msg1:
    .ascii "helloworld, pb16071259_zhangrunbang"
    msg2:
    .ascii "HELLOWORLD, PB16071259_ZHANGRUNBANG"

.text       #代码段
    start:                  #程序的入口点
    movl $0,%edi            #对edi赋初值为0
    movl $35,%ecx           #对ecx赋初值为35
    movw $0x3F8, %dx        #端口地址传给dx

    loop1:                  #进入循环
    movb msg1(,%edi,1), %al     #msg1的字符逐个传给al
    movb %al,0xB8000(,%edi,2)   #字符的值传到VGA显存
    movb $0x2f,0xB8001(,%edi,2) #字符样式传到VGA显存
    movb msg2(,%edi,1), %al     #msg2的字符逐个传给al
    outb %al, %dx           #在端口写入al
    incl %edi               #edi加1
    loop loop1              #跳转回loop1，除非ecx为0

    nop                     #空指令控制时间
    nop
    hlt                     #停止程序运行

    movl $1,%eax
    movl $4,%ebx
    int $0*80               #退出
```

说明：

包含三个段：.multiboot_header段、.data段、.text段。

首先用.section自定义了一个段，段名为.multiboot_header。.multiboot_header段是一个符合multiboot协议的header，用.long声明了三个32位的数，分别为magic、flag、checknum。

.data段用.ascii定义了两个字符串，存入内存便于后面的循环引用。msg1和msg2是两个字符串的符号，代表字符串首个元素的地址。

.text段为代码段，start符号是程序的入口地址，并已在开头用.global声明。

movl $0,%edi

movl $35,%ecx

用mov命令对ecs和edi分别赋初值35和0。ecx寄存器专门存放循环次数，表示有35次循环。edi是一个用于寻址的循环计数。loop1标记了循环的入口地址，执行到loop loop1时ecs的值减1，跳转回loop1，除非ecx为零。

movw $0x3F8, %dx

将端口地址传给dx寄存器。

movb msg1(,%edi,1), %al

从loop1处进入循环后，首先对al传值，msg1(,%edi,1)是一种寻址方式，指向物理地址为msg+edi*1的存储单元，在循环中，该寻址方式顺序遍历了msg1指向的字符串的每个字符，然后用movb将该值传给al。

movb %al,0xB8000(,%edi,2)

al为要传递的字符。将al传给0xB8000(,%edi,2)指向的存储单元。0xB8000(,%edi,2)即0xB8000+edi*2，因为VGA一个字符需要两个字节，所以需要*2.

movb $0x2f,0xB8001(,%edi,2)

含义同上，但传递的是一个固定值2f，指定了字符的显示格式：绿底白字。

movb msg2(,%edi,1), %al

在循环中遍历了msg2指向的字符串的每个字符，并传值给al。

outb %al, %dx

incl %edi

再用out指令在端口中写入al。b表示写入的是单字节。edi的值加1。

nop是空指令，用于控制CPU时间周期，会自动对齐CPU内存寻址。hlt使程序停止运行，处理器进入暂停状态，不执行任何操作。

movl $1,%eax ；movl $4,%ebx ； int $0*80　　用于退出

代码布局：按照右图命令布置

1、start为可执行程序的入口点。

2.首先定位到1M地址处。可执行文件的.text段从此处开始。开始存放所有输入文件 (MultibootHeader.S) 的.multiboot_header段[12字节]，往后对齐8字节后，再存放所有输入文件(MultibootHeader.S)的.text段。

3.接着存放可执行文件的.data段，包括的内容即为所有输入文件(MultibootHeader.S)的.data段。

```
4   ENTRY(start)
5   SECTIONS
6   {
7       . = 1M;
8       .text :
9           {
10          *(.multiboot_header)
11          . = ALIGN(8);
12          *(.text)
13          }
14      .data : { *(.data) }
15  }
```

编译过程：

使用了老师提供的makefile。只需把MultibootHeader.S、MultibootHeader.ld和makefile放在同一文件夹下，终端里make即可。makefile中主要命令有：

gcc -c ${ASM_FLAGS} multibootHeader.S -o multibootHeader.o

用gcc工具把源程序multibootHeader.S编译成目标代码放在目标文件multibootHeader.o中。

ld -n -T multibootHeader.ld multibootHeader.o -o multibootHeader.bin

用ld工具把目标文件multibootHeader.o链接起来得到可执行程序multibootHeader.bin

运行及运行结果：

在包含了multibootHeader.bin的目录下键入下面的命令：

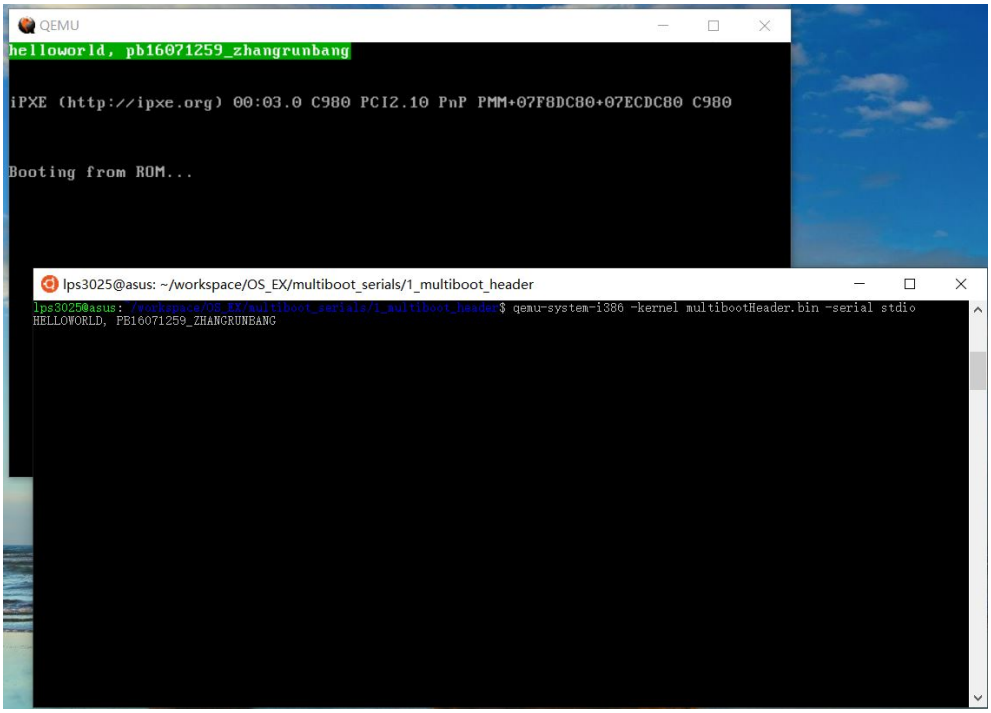
qemu-system-i386 -kernel multibootHeader.bin -serial stdio

（-kernel：指定了内核文件； -serial：指定串口设备，此处使用标准输出）

回车后得到运行结果，如下图所示。包括两方面内容：

SDL窗口显示绿底白字的字符串“helloworld, PB16071259_zhangrunbang”

终端上打印出 "HELLOWORLD, PB16071259_ZHANGRUNBANG"



遇到的问题及解决方案：

一开始不了解汇编，就通过阅读《汇编语言》(第三版，王爽著)了解其语法和背后含义(寄存器概念)，但书上用的不是AT&T的汇编语言格式，所以又参考了https://www.iteye.com/blog/xp9802-2012231 和 https://www.cnblogs.com/orlion/p/5765339.html 两篇文章的内容学习相关汇编指令和伪指令。

由于输出字符数较多，所以增加一个.data段定义字符串，并用循环输出字符。主要查询和学习了汇编中内存寻址的格式。

在语法和环境配置上还碰到了许多小错误，在此感谢助教的纠正和建议。